$1.80

Apple Assembly Line

Volume 5 -- Issue 12 September, 1985

Many of you have added 65802 processors to your Apples, and are now looking for more data on programming the new features of this powerful chip. We've been getting several calls per week asking: "Now that I have this thing, how can I find out more about it?" Well, this issue of AAL will keep you folks busy for a while! We have that standard benchmark, the Sieve of Eratosthenes, coded in 65802, along with a startlingly small routine to convert binary to decimal. And more to come...

In another couple of months there will be a significant addition to the 65816 library. We've been previewing a copy of the galley proofs of a new book on the 65816 by David Eyes. We will have a full review of this important resource, and copies for sale, as soon as the book is really available.

"Now that You Know Apple Assembly Language, What Can You Do with It?" That's the title of a new book written and published by Jules Gilder, a long time contributor to Apple magazines. We'll have a full review next month, and may be carrying the book. In the meantime, see Jules' ad on page 7 of this issue.

S-C Macro Assembler Version 2.0 DOS Source Code

Here's another item we've had many requests for: the source code to S-C Macro Assembler Version 2.0. Now that the DOS version has been out a few months, and seems to be stable, we're releasing the source code. Registered owners of S-C Macro Assembler Version 2.0 can purchase the source, on disk, for only $100. Those of you who purchased the source of an earlier Macro version may add the 2.0 source for only $50. It will be a few more months until the ProDOS Version source code appears.

Prime Benchmark for 65802.....Bob Sander-Cederlof

Jim Gilbreath really started something. He is the one who popularized the use of the Sieve of Eratosthenes as a benchmark program for microcomputers and their various languages. You can read about it in BYTE September 1981, "A High-Level Language Benchmark"; and later in BYTE January 1983, "Eratosthenes Revisited".

In a nutshell, the benchmark creates an array of 8192 bytes, representing the odd numbers from 1 to 16383. The prime numbers in this array are flagged by the program using the Eratosthenes algorithm. All of the times published in the BYTE articles are for ten repetitions of the algorithm.

The second article lists page after page of timing results for various computers and languages. They range from .0078 seconds for an assembly language version running in an IBM 3033, to 5740 seconds for a Cobol version in a Xerox 820.

There are many factors which affect the results, not just the basic speed of the computer involved. The language used is obviously significant, as some languages are more efficient than others for particular purposes. Slight variations in the implementation of the Eratosthenes algorithm can be very significant. The skill and persistence of the programmer are also very important.

Gilbreath's times for the Apple II vary from 2806 seconds for an Applesoft version to 160 seconds for a Pascal version. The same table shows an OSI Superboard, using a 6502 like the Apple, ran an assembly language version in 13.9 seconds. (I don't know what the clock rate of the OSI board was.)

We have published a series of articles in AAL on the same subject. "Sifting Primes Faster and Faster", in October 1981, gave programs in Apple assembly language by William Robert Savoie and myself. At the time I had overlooked the fact that BYTE's times were for ten trips through the program, so I was perhaps a little overly enthusiastic. The table below shows the adjusted times for ten repetitions.

Version	Time in seconds
My Integer BASIC version	1880
Mike Laumers Int BASIC	2380
Mike's compiled by FLASH!	200
Bill Savoie's 6502 assembly	13.9
My first re-write of Bill's	9.3
My 6502 version	7.4
My 6502 with faster clear	6.9

I challenged you readers to do it faster, and some of you did. Charles Putney ("Even Faster Primes", Feb 1982 AAL) knocked the time for ten trips down to 3.3 seconds. Tony Brightwell ("Faster than Charlie", Nov 1982 AAL) combined tricks from number theory with a faster array clear technique to trim the time to 1.83 seconds.

S-C Macro Assembler Version 2.0.....DOS $100, ProDOS $100, both for $120
ProDOS Upgrade Kit for Version 2.0 DOS owners.....$30
Version 2.0 Upgrade Kit for 1.0/1.1/1.2 owners.....$20
Source Code of S-C Macro 2.0 (DOS only).....additional $100
Full Screen Editor for S-C Macro (with complete source code).....$49
S-C Cross Reference Utility.....without source code $20, with source $50
RAK-Ware DISASM.....$30
Source Code for DISASM.....additional $30
S-C Word Processor (with complete source code).....$50
DP18 Source and Object.....$50
Double Precision Floating Point for Applesoft (with source code).....$50
"Bag of Tricks", Worth & Lechner, with diskette.....($39.95) $36
MacASM -- Macro Assembler for MacIntosh (Mainstay).....($150.00) $100
S-C Documentor (complete commented source code of Applesoft ROMs).....$50
Source Code of //e CX & F8 ROMs on disk.....$15
Cross Assemblers for owners of S-C Macro Assembler.....$32.50 to $50 each

(Available: 6800/1/2, 6301, 6805, 6809, 68000, Z-80, Z-8, 8048, 8051, 8085, 1802/4/5, PDP-11, GI1650/70, others)

AAL Quarterly Disks.....each $15, or any four for $45

Each disk contains the source code from three issues of AAL, saving you lots of typing and testing.		Jan-Mar	Apr-Jun	Jul-Sep	Oct-Dec
	1980	-	-	-	1
	1981	2	3	4	5
	1982	6	7	8	9
	1983	10	11	12	13
	1984	14	15	16	17
	1985	18	19	20	

(All source code is formatted for S-C Macro Assembler. Other assemblers require some effort to convert file type and edit directives.)

Diskettes (with hub rings)..... package of 20 for $32
Vinyl disk pages, 6"x8.5", hold two disks each.....10 for $6
Diskette Mailing Protectors (hold 1 or 2 disks).....40 cents each
(Cardboard folders designed to fit 6"X9" Envelopes.) or $25 per 100
Envelopes for Diskette Mailers..... 6 cents each

65802 Microprocessor (Western Design Center).....($95) $50
quikLoader EPROM System (SCRG).....($179) $170
PROmGRAMER (SCRG).....($149.50) $140
Switch-a-Slot (SCRG).....($190) $175
Extend-a-Slot (SCRG).....($35) $32

"Apple ProDOS: Advanced Features for programmers", Little..($17.95) $17
"Inside the Apple //c", Little.....($19.95) $18
"Inside the Apple //e", Little.....($19.95) $18
"Apple II+/IIe Troubleshooting & Repair Guide", Brenner.....($19.95) $18
"Apple][Circuit Description", Gayler.....($22.95) $21
"Understanding the Apple II", Sather.....($22.95) $21
"Understanding the Apple //e", Sather.....($24.95) $23
"Enhancing Your Apple II, vol. 1", Lancaster.....($15.95) $15
"Enhancing Your Apple II, vol. 2", Lancaster.....($17.95) $17
"Assembly Cookbook for the Apple II/IIe", Lancaster.....($21.95) $20
"Beneath Apple DOS", Worth & Lechner.....($19.95) $18
"Beneath Apple ProDOS", Worth & Lechner.....($19.95) $18
"6502 Subroutines", Leventhal.....($18.95) $18
"Real Time Programming -- Neglected Topics", Foster.....($9.95) $9
"Microcomputer Graphics", Myers.....($12.95) $12
"Assem. Language for Applesoft Programmers", Finley & Myers.($16.95) $16
"Assembly Lines -- the Book", Wagner.....($19.95) $18
"AppleVisions", Bishop & Grossberger.....($39.95) $36

Add $1.50 per book for US shipping. Foreign orders add postage needed.
Texas residents please add 6 1/8 % sales tax to all orders.

*** S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 ***
*** (214) 324-2050 ***
*** We accept Master Card, VISA and American Express ***

Peter McInerney sent us an implementation he did on the DTACK Grounded 68000 board, which uses a 12.5 MHz clock. His program ("68000 Sieve Benchmark", July 1984 AAL) did 10 repetitions in .4 seconds. (An 8 MHz time was logged in the BYTE article at .49 seconds. Upping the clock speed does not always speed everything up proportionally, due to the need to wait for slower memory chips.) I translated Peter's code back to 6502 code in "Updating the 6502 Prime Sifter", same issue. My time for ten loops was 1.75 seconds. In that article I stated, "...it remains to be seen what the 65802 could do.

David Eyes, in his new book on 65816 Assembly Language, presents a version which uses the expanded capabilities in that chip. He evidently did not build on our base, because his time for a 4 MHz 65816 was 1.56 seconds. I presume that means if the clock rate was the same as Apple's it would have taken 6.24 seconds. I have been previewing David's book, from the galleys, but the listing of that program was not included in the material I received from the typesetter.

I decided to try updating my 1984 version to 65802 code, using whatever tricks I could come up with. The result runs ten times in 1.4 seconds in the 65802 plugged into my Apple II Plus. I suppose that means a 4 MHz version would run in .35 seconds, or faster than a 12.5 MHz 68000!

Lines 1100-1210 are an outer shell to drive the PRIME program. The shell begins and ends by ringing the Apple bell, to help me run my stopwatch. I ran the PRIME program 1000 times, and then divided the time by 100 to get the seconds for ten repetitions. In between ringing the bells everything is done in 65802 mode. Lines 1110-1120 turn on "native" mode, and lines 1190-1200 restore "emulation" mode.

When you switch on native mode the M and X bits always come up as 1's. That is, both are set to 8-bit mode. The M-bit controls the size of operations on the A-register, and the X-bit controls the size for the X- and Y-registers. Line 1130 turns on 16-bit mode for the A-register. I use this setting throughout the rest of the program, until we go back to emulation mode. All operations which affect the A-register will be 16-bits, while I will only use X and Y with 8-bit values.

Lines 1140-1180 call PRIME 1000 times. Since I have Mbit=0, line 1140 uses the 16-bit LDA immediate. STA COUNT stores both bytes: the low byte at COUNT and and the high byte at COUNT+1. DEC COUNT decrements the full 16-bit value, returning a .NE. status until both bytes are zero. This is certainly a lot easier than a two-byte decrement in 6502 code:

```
        LDA COUNT
        BNE .1
        DEC COUNT+1
  .1    DEC COUNT
        BNE ...         ...NOT AT 0000 YET
        LDA COUNT+1
        BNE ...         ...NOT AT 0000 YET
```

Line 1140 may need some explanation, since there are now at least four assemblers available for the Apple which handle 65802 assembly language. Each of the four have chosen a different way to inform the assembler about the number of bytes to assemble for immediate operands. S-C Macro uses a "#" to indicate and 8-bit operand, and "##" to indicate a 16-bit immediate operand. This seems to me to be the easiest to figure out when I come back to read a program listing after several weeks of working on something else. The "double #" is an immediate visual clue (pun intended) that the immediate operand is double size.

Since ORCA/M was a Hayden Software product, and David Eyes was product manager of ORCA/M at Hayden as well as an early contributor to 65816 design, ORCA/M turned out to be the first assembler to include 65816 support. Mike Westerfield had a version running before the rest of us even knew the 65816 was going to exist. Consequently, Mike's and David's choices for assembly syntax and rules has achieved the honor of being used in the 65816 data sheet and in David's book.

Mike and David decided to inform the assembler what size immediate operands to use with two assembler directives. LONGA controls the size of immediate operands on LDA, CMP, ADC, ORA, EOR, AND, BIT, and SBC: LONGA ON makes them 16-bits, LONGA OFF makes them 8-bits. Likewise, LONGI ON or OFF controls the immediate operands on LDX, LDY, CPX, and CPY. You have to sprinkle your code with these so that the assembler always knows which size to use. Since the directives may not be close to the affected lines of code, it can be a chore to read unfamiliar source code.

Merlin Pro uses a single directive to inform the assembler as to the settings of M and X which will be in effect at execution time. The directive is called "MX", and can have an operand of 0, 1, 2, or 3 (or a symbol whose value is 0-3). The bits of the value correspond to the M- and X-bit settings:

```
MX 0    M=0, X=0  (both 16-bits)
MX 1    M=0, X=1  (A/16, XY/8)
MX 2    M=1, X=0  (A/8, XY/16)
MX 3    M=1, X=1  (A/16, XY/16)
```

I understand that the latest version of Lazerware's Lisa Assembler supports the 65816, but I don't have a copy. I do not know how Randy Hyde indicates immediate operand size.

By the way, in all of the assemblers it is entirely up to the programmer to be sure that you keep all the immediate sizes correct. There is no way for an assembler to second-guess you on this. If you tell it to make a 16-bit operand, and then execute that instruction in 8-bit mode, the third byte will be treated as the next opcode. Vice versa is just as bad. I have blown it many times already, with the result that I am a lot more careful now.

Now let's look at the PRIME subroutine itself. The first section clears an array of 8192 bytes, storing $00 in each

byte. There are a lot of ways to store zeroes. The most obvious is with a loop of STA addr,X lines, such as we used in previous versions. The 65802 has a STZ instruction, which stores zero without using the A-register, but it is not faster. We could store a zero at the beginning of the array and then use an overlapping MVN instruction to copy that zero through the whole array:

```
LDX ##BASE
LDY ##BASE+1
LDA ##8190
MVN 0,0
```

That would be simple, but it would take over 56000 cycles. We can do a lot better than that.

My version uses the PHD instruction 4096 times to push 8192 zeroes on the stack. I start by setting the stack register to point at the last byte of my array (BASE+8191). Each PHD pushes the direct page register (which is currently set to $0000) on the stack. My loop includes 16 PHD's, so 256 times around will fill the array (or empty it, if you like). All this action is in lines 1320-1380. To save space in the source code, rather than write 16 lines of PHD's, I wrote them out as hex strings in lines 1350-1360.

Lines 1310, 1390-1410 save and restore the original stack pointer. (At first I didn't do this, with disastrous results! The stack pointer was sitting just below the cleared array. When I did an RTS, the next opcode encountered was $00, which is a BRK. Since I was in native mode, the BRK vectored through $FFE6,7 instead of $FFFE,F. Et cetera.) Note that the TSX only saves the low byte of S, because X is in 8-bit mode. I am assuming that the high byte was $01, since I came from normal Apple 6502 code. Lines 1390-1400 put $01 in front of the low byte, and the TCS puts both bytes back in the S-register.

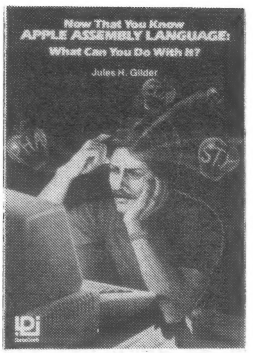
Lines 1430-1440 push the address of the fifth byte in the array onto the stack. Since the 65802 has a stack-relative addressing mode, we can access the pointer with an address of "1,S". Remember the bytes in the array represent the odd numbers. The fifth byte represents the number 9, which is the square of the first odd prime (3). (At a very slight penalty in speed, we can change line 1430 to "LDA ##BASE" and delete line 1460.)

Lines 1480-1520 update the pointer we are keeping on the stack to point to the next square. For an explanation of how this works, go to the July 1984 and Nov 1982 articles. Lines 1530-1540 skip the sifting process for numbers that have already been flagged as non-prime.

Lines 1550-1580 compute the prime number itself from the index (2*index+1) and store it into the operand bytes of the "ADC ##" instruction at line 1630. Ouch! Self-modifying code! But that is often the price of speed.

Line 1590 picks up the pointer to the square of the prime, which is the first number that must be flagged as non-prime,

BECOME AN ASSEMBLY LANGUAGE PROGRAMMING WHIZ



You've spent a lot of time learning Apple assembly language and finally know the difference between BEQ and BCS. Now it's time to put your new-found knowledge to work. Time to throw away your Applesoft programming manual and write programs that make your Apple work like a super-charged, super-fast computer. Time to graduate from the Applesoft BASIC used by beginners, to the 6502 assembly language used by professionals.

To help make this transition, you need an experienced programmer to guide you. You need to develop a library of subroutines that make programming in assembly language as easy as programming in BASIC. You need to learn all the tricks that take experienced assembly language programmers years to acquire. Most important of all, you need the book, *"Now that You Know Apple Assembly Language: What Can You Do With It?"* because it contains all this information *and more.*

It shows you how, step-by-step

"Now That You Know Apple Assembly Language: What Can You Do With It?" will take you step-by-step through the assembly language programming experience. You'll delve into the mysteries of the 6502 stack and learn how to use it to increase the power and versatility of your programs. You'll also learn how to use the Apple's built-in routines to minimize the amount of coding you must do.

Control the output and the input

Frequently it's desirable to gain total control of the computer's output. This book shows you how to *steal control away from the Apple's normal output routines and redirect it to your own program.* Thus if you wanted, you could see the normally invisible control characters, display text on your screen as black on white instead of the normal white on black, format text sent to a printer into pages and much more.

Expand the power of your Apple by *stealing control away from the normal input routines.* Do things like adding a screen print capability, or *convert part of the normal keyboard into a numeric keypad.* It's even possible to *produce self-modifying programs* by EXECing in commands from RAM instead of from the disk drive. Think about the possibilities that offers for protecting your programs. When you want to go back to Applesoft programming, *you'll be able to do it faster with the aid of Applesoft Shorthand,* an assembly language program that types in one or more Applesoft commands at the press of a key, or use another program in the book to *automatically count the number of lines in your Applesoft program.*

With this book you'll also learn about *generating tones and how to figure out the frequency, producing sound effects, teaching your Apple to send Morse code, restoring accidentally erased Applesoft programs, adding new commands to Applesoft and running two Applesoft programs in memory together,* to name a few.

*NOTE: Shipping and handling fees are not refundable.

Everything is explained

Unlike other books that merely consist of a collection of programs, this one explains what's happening, where and why. You get detailed descriptions of how the programs work and detailed program listings with virtually every line of code explained. Nothing is left to chance or misinterpretation.

Order now, get 2 FREE gifts

The book costs only $19.95 plus $2 for shipping and handling. Order now and you'll also get a *FREE Programmer's Number Conversion System* that makes it easy to convert between binary, hexadecimal and decimal numbers. No calculators are required. You'll convert numbers almost instantly and wonder how you ever got along without it.

As an extra bonus for prompt ordering, you'll receive a *FREE coupon worth $5 off* the price of a disk with all the assembled programs on it or a disk that contains the source code. These disks normally sell for $15 each. We're offering these FREE gifts for a limited time only, so hurry! *Order today!*

Money-back guarantee*

We're so confident that you'll find this book invaluable and want it in your library, that we're offering a 10-day, no-questions-asked, money-back guarantee. Order the book. Read it and try the programs for ten days. At the end of ten days if you don't think it's worth every penny you paid for it, just send it back in resalable condition and we'll refund your money immediately, no questions asked.

CALL TOLL FREE 800-235-6646 (In California 800-235-6647) Ext. 674 and use your American Express, Visa or MasterCard. These phone numbers are for credit card orders only. For inquiries call 718-232-8429.

DataCraft, Inc., Dept. A13
2068–79th St., Brooklyn, NY 11214

Please rush me _____ copies of **"Now That You Know Apple Assembly Language: What Can You Do With It?"** at $19.95 each plus $2 shipping and handling. I understand that if I am not delighted with the book I may return it within 10 days for a prompt and courteous refund. In any case, the Programmer's Number Conversion System and $5 coupon are mine to keep.

☐ Enclosed is my check for $ _____

Please charge my credit card:
☐ American Express ☐ MasterCard ☐ Visa

Card No. _____ Exp. _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

from our holding location on the stack. Lines 1610-1640 get tricky. Line 1610 puts the current pointer in the D-register, which tells where in RAM the direct page starts. This means that the "STX 0" in line 1620 stores into the byte pointed to. X was holding the current index, so we are storing a non-zero number into that byte, which flags it as being non-prime.

As a pleasant side effect, the non-zero numbers being stored in the array have meaning. If we double the value we stored and add one, we will get the value of the prime factor of the non-prime number. After the whole PRIME program has executed, the flag value will produce the largest prime factor.

In the loop of lines 1610-1640, we keep adding the prime number to the pointer value in the A-register, and transferring the result to the D-register. Hence the STX 0 will store X at multiples of the prime number. The loop terminates when the pointer value in the A-register goes negative. Why? Because we carefully positioned the array from $6000 to $7FFF. The first time we add the prime to the pointer and get an address $8000 or higher, we know we went off the end of the array. Addresses of $8000 or higher will set the negative status flag, so our loop terminates.

Lines 1660-1680 bump the prime index by one, and test for having reached the largest prime of interest. If not, we go back to sift out the next one. If we are finished, lines 1690-1700 restore the D-register to point to true page zero. Line 1710 pops the pointer off the stack, and that's all there is to it!

```
                        1000          .OP 65816
                        1010 *SAVE S.SUPER-FAST PRIMES 65802
                        1020          .OR $8000     SAFELY OUT OF WAY
                        1030 *--------------------------------
6000-                   1040 BASE     .EQ $6000     PRIME ARRAY $6000...7FFF
FF3A-                   1050 BEEP     .EQ $FF3A     BEEP THE SPEAKER
00-                     1060 COUNT    .EQ 0,1
                        1070 *--------------------------------
                        1080 *        MAIN CALLING ROUTINE
                        1090 *--------------------------------
008000- 20 3A FF        1100 MAIN     JSR BEEP
008003- 18              1110          CLC           ...ENTER NATIVE MODE
008004- FB              1120          XCE
008005- C2 20           1130          REP #$20      A/16, XY/8
008007- A9 E8 03        1140          LDA ##1000      DO IT 1000 TIMES
00800A- 85 00           1150          STA COUNT
00800C- 20 18 80        1160 .1       JSR PRIME
00800F- C6 00           1170          DEC COUNT
008011- D0 F9           1180          BNE .1
008013- 38              1190          SEC           ...ENTER EMULATION MODE
008014- FB              1200          XCE
008015- 4C 3A FF        1210          JMP BEEP      SAY WE'RE DONE
                        1220 *--------------------------------
                        1230 *        PRIME ROUTINE
                        1290 *--------------------------------
                        1300 PRIME
008018- BA              1310          TSX           SAVE STACK PNTR
008019- A0 00           1320          LDY #0        256 * 16 * 2 = 8192 BYTES
00801B- A9 FF 7F        1330          LDA ##BASE+8191   BASE...BASE+8191
00801E- 1B              1340          TCS           TEMPORARY STACK PNTR
00801F- 0B 0B 0B 0B
008023- 0B 0B 0B 0B     1350 .1       .HS 0B.0B.0B.0B.0B.0B.0B.0B  ...16 PHD'S
```

```
008027- 0B 0B 0B 0B
00802B- 0B 0B 0B 0B 1360         .HS 0B.0B.0B.0B.0B.0B.0B.0B
00802F- 88          1370         DEY             256 TIMES
008030- D0 ED       1380         BNE .1
008032- 8A          1390         TXA
008033- 09 00 01    1400         ORA ##$0100     RESTORE STACK PNTR
008036- 1B          1410         TCS
                    1420 *-------------------------------
008037- A9 04 60    1430         LDA ##BASE+4    POINT AT FIRST PRIME-SQUARED
00803A- 48          1440         PHA                 (WHICH IS 3*3=9)
00803B- A2 01       1450         LDX #1          POINT AT FIRST PRIME (3)
00803D- D0 0C       1460         BNE .4          ...ALWAYS
                    1470 *-------------------------------
00803F- 8A          1480 .2      TXA
008040- 0A          1490         ASL
008041- 0A          1500         ASL             *4, CLEARS CARRY TOO
008042- 63 01       1510         ADC 1,S         ADD TO PREVIOUS PNTR
008044- 83 01       1520         STA 1,S         PNTR TO SQUARE OF ODD NUMBER
008046- BC 00 60    1530         LDY BASE,X      GET A POSSIBLE PRIME
008049- D0 10       1540         BNE .8          THIS ONE HAS BEEN KNOCKED OUT
00804B- 8A          1550 .4      TXA
00804C- 0A          1560         ASL             DELTA = START*2 + 1
00804D- 1A          1570         INC
00804E- 8D 57 80    1580         STA .7+1
008051- A3 01       1590         LDA 1,S         PNTR TO SQUARE OF PRIME
                    1600 *---STRIKE OUT MULTIPLES---------
008053- 5B          1610 .6      TCD             POINT AT MULTIPLE
008054- 86 00       1620         STX 0           STORE NON-ZERO AS FLAG
008056- 69 00 00    1630 .7      ADC ##*-*       (VALUE FILLED IN)
008059- 10 F8       1640         BPL .6          ...NOT FINISHED
                    1650 *---NEXT ODD NUMBER--------------
00805B- E8          1660 .8      INX
00805C- E0 40       1670         CPX #64         UP TO 127
00805E- 90 DF       1680         BCC .2          WE'RE DONE IF X>127
008060- A9 00 00    1690         LDA ##0         RESTORE DIRECT PAGE REGISTER
008063- 5B          1700         TCD
008064- 68          1710         PLA             POP PNTR OFF STACK
008065- 60          1720         RTS
                    1730 *-------------------------------
```

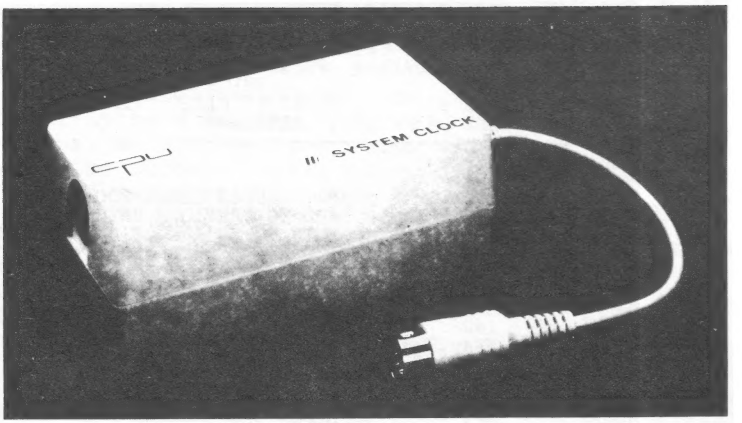
Here is an Applesoft program which will look through the array PRIME produces. Every zero byte in the array indicates a prime number. The value of the prime number at ARRAY+I is I*2+1, since the array only represents odd numbers. This program prints out the value 1 first, which really is not considered a prime number, but it does make the table easier to read.

The program is designed to display 10 8-character fields on a line, which works well on the Apple 80-column screen. I left out the code to print a RETURN after 10 numbers, because the Apple screen automatically goes to the next line.

Line 120 prints out the primes. Delete line 125 if all you want to see is primes. Line 125 prints the largest prime factor of nonprimes, followed by "*" and the other factor (which may not be prime). For example, 16383 is printed as 127*129.

```
100  HIMEM: 24576
110  FOR A = 24576 TO 32767
120  IF  PEEK (A) = 0 THEN
      PRINT RIGHT$("       " + STR$((A - 24576)*2+1),8);
125  IF  PEEK (A) <> 0 THEN
      F1 =  PEEK (A) * 2 + 1
      : F2 = ((A - 24576) * 2 + 1) / F1
      : PRINT RIGHT$("      "+STR$(F1)+"*"+STR$(F2),8);
140  NEXT
```

NEW PRODUCT



IIc SYSTEM CLOCK

- Fully ProDos compatible
- Automatic time and date stamping
- Easy to use from BASIC
- Battery operated, uses 3 "AA" batteries (will last 1-2 years before simple replacement)
- Date has year, month, date and day of week
- Time has hours, minutes and seconds
- Will time and date stamp AppleWorks files
- Will display time and date on the AppleWorks screen
- Auto access from AppleWorks data-base (just use a time and date field)
- Pass through serial port - The IIc system clock can plug into either the modem or printer serial port, then modem or printer plugs into the clock
- No hassle 5 year warranty
- Only $79.00

AE

"We Set the Standard"

214-241-6060 APPLIED ENGINEERING 9 AM - 11 PM

Put DOS and ProDOS Files on Same Disk.....Bob Sander-Cederlof

In the February 1985 issue of AAL I showed how to create a DOS-less DOS 3.3 data disk. Tracks 1 and 2, normally full of the DOS image, were instead made available for files. Booting the disk gets you a message that such a disk cannot be booted.

Now that we are publishing more and more programs intended for use under ProDOS, we foresee the need to publish Quarterly Disks that contain both DOS and ProDOS programs. Believe it or not, this is really possible.

The DOS operating system keeps its Volume Table of Contents (VTOC) and catalog in track $11. The VTOC is in sector 0 of that track, and the catalog normally fills the rest of the track. A major part of the VTOC is the bit map, which shows which sectors are as yet unused by any files. If we want to reserve some sectors for use by a ProDOS directory on the same disk, we merely mark those sectors as already being in use in the DOS bit map.

ProDOS keeps its directory and bit map all in track 0. This track is not available to DOS for file storage anyway, so we can be comfortable stealing it for a ProDOS setup on the same diskette.

I decided to keep things fairly simple, by splitting the disk into two parts purely on a track basis. ProDOS gets some number of tracks starting with track 0, and DOS gets all the tracks from just after ProDOS to track 34. If ProDOS gets more than 17 tracks, it will hop over track $11 (since DOS's catalog is there). Normally I will split the disk in half, giving tracks 0-16 to ProDOS and tracks 17-34 to DOS. With this arrangement, ProDOS thus starts with 129 free blocks, and DOS starts with 272 free sectors.

The program I wrote does not interact with the user; instead, you set all the options by changing the source code and re-assembling. It would be nice to have an interactive front end to get slot, drive, volume number for the DOS half, volume name for the ProDOS half, and how many tracks to put in each half. Maybe we'll add this stuff later, or maybe you would like to try your hand at it.

The parameters you might want to change are found in lines 1020-1050. You can see that I started the DOS allocation at track $12, just after the catalog track. I also chose volume 1, drive 1, slot 6. You can use any volume number from 1 to 254. Since these numbers were under my control, I did not bother to check for legal values. If we add an interactive front end, we will have to validate them. We might also want to display the number of ProDOS blocks and DOS sectors that result from the DOS.LOW.TRACK selection, maybe in a graphic format. You might even use a joystick or mouse....

You might also want to change the ProDOS volume name. I am calling it "DATA". The name is in line 2850. It can be up to 15 characters long, and the number of characters must be stored

in the right nybble of the byte just before the name. This is automatically inserted for you, by the assembler. If you should try to assemble a name larger than 15 characters, line 2870 will cause a RANGE ERROR. Another way of changing the ProDOS volume name is to do so after initialization using the ProDOS FILER program.

Lines 1090 and 1100 compute the number of free DOS sectors and ProDOS blocks. The values are not used anywhere in the program, but are nice to know.

Line 1300 sets the program origin at $803. Why $803, and not $800? If we load and run an assembly language program at $800, and then later try to load and run an Applesoft program, Applesoft can get confused. Applesoft requires that $800 contain a $00 value, but it does not make sure it happens when you LOAD an Applesoft program from the disk. By putting our program at $803 we make sure we don't kill the $00 and $800. Well, then why not start at $801? I don't know, we just always did it that way. (It would make good sense if our program started by putting $00 in $801 and $802, indicating to Applesoft that it had no program in memory.)

DOUBLE.INIT is written to run under DOS 3.3, and makes calls on the RWTS subroutine to format and write information on the disk. The entire DOUBLE.INIT program is driven by lines 1320-1490. The flow is very straightforward:

1. Format the disk as 35 empty tracks.
2. Write DOS VTOC and Catalog in track 17.
3. Write ProDOS Directory and bit map in track 0.
4. Write "YOU CANNOT BOOT" code in boot sector.

Formatting a blank disk is simple, unless you have a modified DOS with the INIT capability removed. Lines 1510-1590 set up a format call to RWTS, and fall into my RWTS caller.

Lines 1600-1800 call RWTS and return, unless there was an error condition. If there was an error, I will print out "RWTS ERROR" and the error code in hex. The error code values you might see are:

$08 -- Error during formatting
$10 -- Trying to write on write protected disk
$40 -- Drive error

I don't think you can get $20 (volume mismatch) or $80 (read error) from DOUBLE.INIT. After printing the error message, DOS will be warm started, aborting DOUBLE.INIT.

Building the DOS VTOC and Catalog is handled by lines 1820-2310. The beginning section of the VTOC contains information about the number of tracks and sectors, where to find the catalog, etc. This is all assembled in at lines 2260-2310, and is copied into my buffer by lines 1880-1930. Since the volume number is a parameter, I specially load it in with lines 1940 and 1950. The rest of the VTOC is a bit map showing which sectors are not yet used. Lines 1960-2090 build this bit map.

Lines 1840-1870 and 2100-2120 cause the VTOC image to be written on track 17 ($11) sector 0.

There are some unused bytes in the beginning part of the VTOC, so I decided to put some private information in there. See line 2270 and line 2290.

The rest of track 17 is a series of empty linked sectors comprising the catalog. The chain starts with sector $0F, and works backward to sector 1. Lines 2130-2240 build each sector in turn and write it on the disk.

The ProDOS directory and bit map are installed in track 0 by lines 2330-2900. This gets a little tricky, because we are trying to write ProDOS blocks with DOS 3.3 RWTS. Here is a correspondence table, showing the blocks and sectors in track 0:

```
    ProDOS Block:  0   1   2   3   4   5   6   7
DOS 3.3 Sectors: 0,E D,C B,A 9,8 7,6 5,4 3,2 F,1
```

The first sector of each pair contains the first part of each block, and so on.

The ProDOS bit map goes in block 6, which is sectors 3 and 2. Even if we had an entire diskette allocated to ProDOS the bit map would occupy very little of the first of these two sectors. Since formatting the disk wrote 256 zeroes into every sector, we can leave sector 2 unchanged. Lines 2700-2820 build the bit map data for sector 3 and write it out. Note that block 7 is available, all blocks in track 17 are unavailable.

The ProDOS Directory starts in block 2. The first two bytes of a directory sector point to the previous block in the directory chain, and the next two bytes point to the following block in the chain. We follow the standard ProDOS convention of linking blocks 3, 4, and 5 into the directory. Those three blocks contain no other information, since there are as yet no filenames in the directory. Here's how the chain links together:

	Previous Block	Next Block	
Block 2:	0	3	(zero means the beginning)
Block 3:	2	4	
Block 4:	3	5	
Block 5:	4	0	(zero means the end)

Block 2 gets some extra information, the volume header. Lines 2840-2900 contain the header data, which is copied into my buffer by lines 2590-2630.

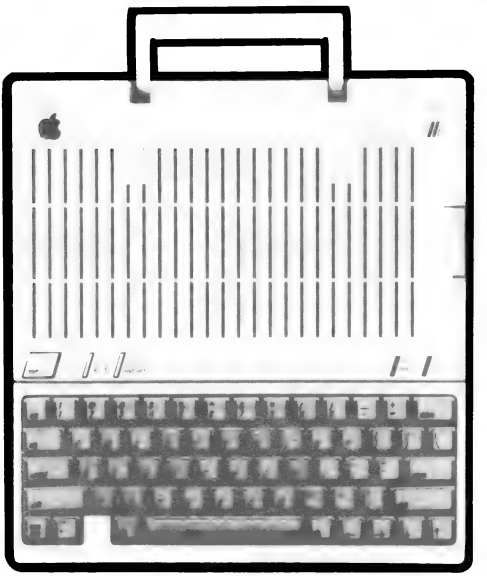
The no-booting boot program is shown in lines 3000-3190. This is coded as a .PHase at $800 (see lines 3010 and 3190), since the disk controller boot ROM will load it at that address. All the program does is turn off the disk motor and print out a little message. Lines 1410-1490 write this program on track 0 sector 0.

I think if you really wanted to you could put a copy of the ProDOS boot program in block 0 (sectors 0 and E). Then if you copied the file named PRODOS into the ProDOS half of the disk, you could boot ProDOS.

There is one thing to look out for if you start cranking out DOUBLE DISKS. There are some utility programs in existence which are designed to "correct" the DOS bitmap in the VTOC sector. Since these programs have never heard of ProDOS, let alone of DOUBLE DISKS, they are going to tell DOS that all those tracks we carefully gave to ProDOS belong to DOS. If you let that happen to a disk on which you have already stored some ProDOS files, zzzaaaapppp!

```
                    1000 *SAVE S.INIT DOS & PRODOS
                    1010 *--------------------------------
12-                 1020 DOS.LOW.TRACK .EQ $12    DOS $12...$22
01-                 1030 DOS.VOLUME    .EQ 1
06-                 1040 SLOT          .EQ 6
01-                 1050 DRIVE         .EQ 1
                    1060 *--------------------------------
90-                 1070 PRODOS.MAX.BLOCKS .EQ DOS.LOW.TRACK*8
                    1080 *--------------------------------
0110-               1090 ACTUAL.DOS.SECTORS   .EQ DOS.LOW.TRACK>$11
                                                    +34-DOS.LOW.TRACK*16
81-                 1100 ACTUAL.PRODOS.BLOCKS .EQ DOS.LOW.TRACK<$12
                                                    +DOS.LOW.TRACK-2*8+1
                    1110 *--------------------------------
03D0-               1120 DOS.WARM.START .EQ $03D0
03D9-               1130 RWTS          .EQ $03D9
03E3-               1140 GETIOB        .EQ $03E3
                    1150 *--------------------------------
B7E8-               1160 R.PARMS       .EQ $B7E8
B7E9-               1170 R.SLOT16      .EQ $B7E9
B7EA-               1180 R.DRIVE       .EQ $B7EA
B7EB-               1190 R.VOLUME      .EQ $B7EB
B7EC-               1200 R.TRACK       .EQ $B7EC
B7ED-               1210 R.SECTOR      .EQ $B7ED
B7F0-               1220 R.BUFFER      .EQ $B7F0,B7F1
B7F4-               1230 R.OPCODE      .EQ $B7F4
B7F5-               1240 R.ERROR       .EQ $B7F5
                    1250 *--------------------------------
FD8E-               1260 MON.CROUT     .EQ $FD8E
FDDA-               1270 MON.PRBYTE    .EQ $FDDA
FDED-               1280 MON.COUT      .EQ $FDED
                    1290 *--------------------------------
                    1300         .OR $803
                    1310 *--------------------------------
                    1320 DOUBLE.INIT
0803- 20 2E 08      1330         JSR FORMAT.35.TRACKS
0806- A9 FC         1340         LDA #INIT.BUFFER
0808- 8D F0 B7      1350         STA R.BUFFER
080B- A9 09         1360         LDA /INIT.BUFFER
080D- 8D F1 B7      1370         STA R.BUFFER+1
0810- 20 75 08      1380         JSR BUILD.DOS.CATALOG
0813- 20 01 09      1390         JSR BUILD.PRODOS.CATALOG
                    1400 *---WRITE BOOT PROGRAM-----------
0816- A9 A4         1410         LDA #BOOTER
0818- 8D F0 B7      1420         STA R.BUFFER
081B- A9 09         1430         LDA /BOOTER
081D- 8D F1 B7      1440         STA R.BUFFER+1
0820- 20 9A 09      1450         JSR CLEAR.INIT.BUFFER
0823- A9 00         1460         LDA #0
0825- 8D EC B7      1470         STA R.TRACK
0828- 8D ED B7      1480         STA R.SECTOR
082B- 4C 45 08      1490         JMP CALL.RWTS
                    1500 *--------------------------------
```

640K



Fatter than a Fat Mac and ready for serious business — that's an Apple //c with a MultiRam™ C Card installed. MultiRam adds from 256K to 512K to your //c's own 128K memory to give you up to a 640K //c.

With a 640K Apple //c you can...

- Run AppleWorks with a 413K Desktop. That's a Desktop large enough for the most demanding business or personal use. Data bases and spreadsheets can grow beyond the frustrating limitations of the old 55K Desktop.

  You can even eliminate AppleWork's need for an external disk drive and speed up AppleWorks by loading all the program disk into MultiRam's memory.

- Run the new SuperCalc3[a], an integrated spreadsheet for your //c with the power of Lotus 1-2-3, and create complex models up to 512K in size.

- Run FlashCalc, Magic Office, MagicCalc and future programs that will use most or all of MultiRam's memory.

- Use MultiRam C's memory as a RAM disk for ProDOS and DOS equaling up to four standard //c disk drives with speeds more than 20 times normal.

The MultiRam C Card...

- Easily installs inside the //c using only a screwdriver. No messy soldering or jumpers are required. MultiRam's expandable memory is easily increased if you choose a 256K version.

- Includes a comprehensive manual, AppleWorks memory expansion software, ProDOS and DOS RAM disk software, and RAM test software at no additional charge.

- Features a full 5-year warranty and is 100% compatible with all //c hardware and software. MultiRam uses a low 1.2 watts of power with its CMOS power saving design. CMOS insures longer sessions when using a battery pack with your //c.

Available at a ***DISCOUNT*** from:

Coit Valley Computers
14055 Waterfall Way, Dallas, Texas 75240
214-234-5047

Checkmate Technology, Inc.

509 South Rockford Drive • Tempe, Arizona 85281-3021
(602) 966-5802 U.S.A. Telex: 165-025 CEC PHX

```
                        1510 FORMAT.35.TRACKS
082E- A9 60             1520         LDA #SLOT*16
0830- 8D E9 B7          1530         STA R.SLOT16
0833- A9 01             1540         LDA #DRIVE
0835- 8D EA B7          1550         STA R.DRIVE
0838- A9 01             1560         LDA #DOS.VOLUME
083A- 8D EB B7          1570         STA R.VOLUME
083D- 8D 02 0A          1580         STA V.VOLUME
0840- A9 04             1590         LDA #$04      INIT OPCODE FOR RWTS
                        1600 CALL.RWTS.OP.IN.A
0842- 8D F4 B7          1610         STA R.OPCODE
                        1620 CALL.RWTS
0845- 20 E3 03          1630         JSR GETIOB
0848- 20 D9 03          1640         JSR RWTS
084B- B0 01             1650         BCS .1        ERROR
084D- 60                1660         RTS
084E- A0 00             1670 .1      LDY #0        PRINT "ERROR"
0850- B9 67 08          1680 .2      LDA ERMSG,Y
0853- F0 06             1690         BEQ .3
0855- 20 ED FD          1700         JSR MON.COUT
0858- C8                1710         INY
0859- D0 F5             1720         BNE .2        ...ALWAYS
085B- AD F5 B7          1730 .3      LDA R.ERROR   GET ERROR CODE
085E- 20 DA FD          1740         JSR MON.PRBYTE
0861- 20 8E FD          1750         JSR MON.CROUT
0864- 4C D0 03          1760         JMP DOS.WARM.START
                        1770 *--------------------------------
0867- 8D 87             1780 ERMSG   .HS 8D87
0869- D2 D7 D4
086C- D3 A0 C5
086F- D2 D2 CF
0872- D2 A0             1790         .AS -/RWTS ERROR /
0874- 00                1800         .HS 00
                        1810 *--------------------------------
                        1820 BUILD.DOS.CATALOG
0875- 20 9A 09          1830         JSR CLEAR.INIT.BUFFER
0878- A9 11             1840         LDA #17
087A- 8D EC B7          1850         STA R.TRACK
087D- A9 00             1860         LDA #0
087F- 8D ED B7          1870         STA R.SECTOR
                        1880 *---BUILD GENERIC VTOC-----------
0882- A0 37             1890         LDY #VTOC.SZ-1
0884- B9 C9 08          1900 .0      LDA VTOC,Y
0887- 99 FC 09          1910         STA INIT.BUFFER,Y
088A- 88                1920         DEY
088B- 10 F7             1930         BPL .0
088D- A9 01             1940         LDA #DOS.VOLUME
088F- 8D 02 0A          1950         STA V.VOLUME
                        1960 *---PREPARE BITMAP---------------
0892- A0 88             1970         LDY #4*34
0894- A9 FF             1980         LDA #$FF
0896- C0 44             1990 .1      CPY #4*17     ARE WE ON CATALOG TRACK?
0898- F0 0A             2000         BEQ .2
089A- C0 48             2010         CPY #4*DOS.LOW.TRACK
089C- 90 0C             2020         BCC .3              IN PRODOS ARENA
089E- 99 35 0A          2030         STA V.BITMAP+1,Y
08A1- 99 34 0A          2040         STA V.BITMAP,Y
08A4- 88                2050 .2      DEY
08A5- 88                2060         DEY
08A6- 88                2070         DEY
08A7- 88                2080         DEY
08A8- D0 EC             2090         BNE .1
                        2100 *---WRITE VTOC ON NEW DISK-------
08AA- A9 02             2110 .3      LDA #2              RWTS WRITE OPCODE
08AC- 20 42 08          2120         JSR CALL.RWTS.OP.IN.A
                        2130 *---WRITE CATALOG CHAIN----------
08AF- 20 9A 09          2140         JSR CLEAR.INIT.BUFFER
08B2- A9 11             2150         LDA #17       TRACK 17
08B4- A0 0F             2160         LDY #15       START IN SECTOR 15
08B6- 8D FD 09          2170         STA C.TRACK
08B9- 8C ED B7          2180 .4      STY R.SECTOR
08BC- 88                2190         DEY
08BD- 8C FE 09          2200         STY C.SECTOR
08C0- 20 45 08          2210         JSR CALL.RWTS
08C3- AC FE 09          2220         LDY C.SECTOR
08C6- D0 F1             2230         BNE .4
08C8- 60                2240         RTS
```

```
                        2250 *-------------------------------
08C9- 04 11 0F
08CC- 03 00 00
08CF- 01                2260 VTOC   .HS 04.11.0F.03.00.00.01
08D0- 43 4F 4D
08D3- 42 49 4E
08D6- 41 54 49
08D9- 4F 4E 20
08DC- 44 4F 53
08DF- 2F 50 52
08E2- 4F 44 4F
08E5- 53 20 44
08E8- 41 54 41
08EB- 20 44 49
08EE- 53 4B             2270        .AS "COMBINATION DOS/PRODOS DATA DISK"
08F0- 7A                2280        .HS 7A
08F1- 30 37 2D
08F4- 32 35 2D
08F7- 38 35             2290        .AS /07-25-85/
08F9- 11 01 00
08FC- 00 23 10
08FF- 00 01             2300        .HS 11.01.00.00.23.10.00.01
38-                     2310 VTOC.SZ .EQ *-VTOC
                        2320 *-------------------------------
                        2330 BUILD.PRODOS.CATALOG
0901- A9 00             2340        LDA #0
0903- 8D EC B7          2350        STA R.TRACK
0906- 20 9A 09          2360        JSR CLEAR.INIT.BUFFER
                        2370 *-------------------------------
0909- A9 05             2380        LDA #5              SECTOR 5 = BLOCK 5
090B- 8D ED B7          2390        STA R.SECTOR        BACK LINK = 0004
090E- A9 04             2400        LDA #4               FWD LINK = 0000
0910- 8D FC 09          2410        STA INIT.BUFFER
0913- 20 45 08          2420        JSR CALL.RWTS
                        2430 *-------------------------------
0916- A9 07             2440        LDA #7              SECTOR 7 = BLOCK 4
0918- 8D ED B7          2450        STA R.SECTOR        BACK LINK = 0003
091B- CE FC 09          2460        DEC INIT.BUFFER      FWD LINK = 0005
091E- A9 05             2470        LDA #5
0920- 8D FE 09          2480        STA INIT.BUFFER+2
0923- 20 45 08          2490        JSR CALL.RWTS
                        2500 *-------------------------------
0926- A9 09             2510        LDA #9              SECTOR 9 = BLOCK 3
0928- 8D ED B7          2520        STA R.SECTOR        BACK LINK = 0002
092B- CE FC 09          2530        DEC INIT.BUFFER      FWD LINK = 0004
092E- CE FE 09          2540        DEC INIT.BUFFER+2
0931- 20 45 08          2550        JSR CALL.RWTS
                        2560 *-------------------------------
0934- A9 0B             2570        LDA #11             SECTOR 11 = BLOCK 2
0936- 8D ED B7          2580        STA R.SECTOR        BACK LINK = 0000
0939- A0 2A             2590        LDY #HDR.SZ-1        FWD LINK = 0003
093B- B9 6F 09          2600 .1     LDA HEADER,Y
093E- 99 FC 09          2610        STA INIT.BUFFER,Y GET VOLUME HEADER
0941- 88                2620        DEY
0942- 10 F7             2630        BPL .1
0944- A9 90             2640        LDA #PRODOS.MAX.BLOCKS
0946- 8D 25 0A          2650        STA INIT.BUFFER+$29
0949- A9 00             2660        LDA /PRODOS.MAX.BLOCKS
094B- 8D 26 0A          2670        STA INIT.BUFFER+$2A
094E- 20 45 08          2680        JSR CALL.RWTS
                        2690 *-------------------------------
0951- A9 03             2700        LDA #3
0953- 8D ED B7          2710        STA R.SECTOR
0956- 20 9A 09          2720        JSR CLEAR.INIT.BUFFER
0959- A9 FF             2730        LDA #$FF
095B- A0 11             2740        LDY #DOS.LOW.TRACK-1
095D- C0 11             2750 .2     CPY #17       SKIP OVER DOS CATALOG TRACK
095F- F0 03             2760        BEQ .3
0961- 99 FC 09          2770        STA INIT.BUFFER,Y
0964- 88                2780 .3     DEY
0965- 10 F6             2790        BPL .2
0967- A9 01             2800        LDA #1        MAKE ONLY BLOCK 7 AVAILABLE
0969- 8D FC 09          2810        STA INIT.BUFFER   IN TRACK 0
096C- 4C 45 08          2820        JMP CALL.RWTS
```

12 Good Reasons Why RAMWORKS™ Is The Best Expansion Card For Your IIe

1 APPLEWORKS MEMORY Even though Ramworks enhances and expands a VAST ARRAY of other programs, Appleworks is our claim to fame. A 64K Ramworks will ADD 46K to your available desktop memory, a 128K Ramworks will ADD 91K, a 256K Ramworks will ADD 182K, and a 512K Ramworks will ADD 364K and a 1 meg Ramworks will give you nearly an 800K desktop. And it's all done automatically! When you plug in more memory chips into your Ramworks card, Appleworks will find them—automatically. Ramworks also increases the maximum number of records from 1350 to 4300.

2 APPLEWORKS SPEED AND POWER Ramworks does more than just increase the desktop memory (as if that weren't enough). With Ramworks, Appleworks will be able to run up to 20 times faster. If you buy a 256K or larger Ramworks card, Appleworks will automatically load itself in Ramworks. This greatly increases the speed at which Appleworks operates by eliminating all that nasty, time consuming disk access on Drive 1. These are but a few reasons why we say that Ramworks is Appleworks best friend.

3 EXPANDABILITY Ramworks was designed with the future in mind, as your needs increase, so can Ramworks. Clear instructions show you how to plug in more memory (up to 1 meg).

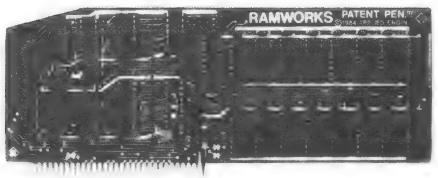
4 SPEED Today, as programs become more and more sophisticated, they inevitably become larger. And many of today's best selling programs (like Appleworks) won't fit in a 128K Apple, so many of these new larger programs continually go back to disk in search of more data. With Ramworks, you can have enough memory so that the entire program will be loaded into Ramworks' memory. This greatly increases the speed of software because your disk runs at 300 RPM, but Ramworks operates at the speed of light!

5 COLOR The same slot that's used for memory expansion is also the slot that's used for RGB color display, so all those lesser memory cards of yesterday make you decide in advance if you want RGB color. Only Ramworks lets you decide later to add RGB color. For only $129, an RGB option can be added to Ramworks to give you double high resolution color graphics and 80 column text. All with a razor sharp, vivid brilliance that's unsurpassed in the industry. The RGB option does not waste another valuable slot, but rather plugs into the back of Ramworks and attaches to any Apple compatible monitor. Remember, you can order the RGB option with your Ramworks or add it on at a later date.

6 COMPATIBILITY, OF THE SOFTWARE KIND Programs like Appleworks, Magic Office System, Flashcalc, The Spread Sheet, Diverse-A-Dos, Supercalc, Magicalc and many others automatically recognize all or most of Ramworks memory (512K is average). The simple fact is that Ramworks is compatible with more off-the-shelf software than any other RAM card. Ramworks is 100% compatible with ALL software written for the Apple 80 column and extended 80 column card. Additionally, Ramworks can emulate other RAM cards so software written for other cards will run without modification. Software written for RAMWORKS will not work on other cards. We can emulate others, but others can't emulate us.

7 COMPATIBILITY, OF THE HARDWARE KIND Unlike others, Ramworks is fully compatible with hardware add on's from other companies, like the Sider and Profile hard disks. And Ramworks was designed in accordance with the official expansion rules defined by Apple so you don't have to worry about compatibility problems. As you continue to expand and make your Apple more powerful with other expansion products from Applied Engineering, you'll appreciate how each product has extra features designed to work with Ramworks and other products to give you a total performance package that is more powerful than the sum of its parts.

8 IT SELLS THE MOST Popularity translates into great software support because software companies can't support all RAM cards, they can only support the ones their customers are likely to own. And software companies appreciate the fact that when they write software for Ramworks in the IIe,



they're also writing software for our memory expansion card for the IIe, Z-RAM. And our customer list reads like the Who's Who of Apple computing with just about every software company in the land buying one, including Apple Computer (in the hundreds), Rupert Lissner, and Steve Wozniak (we didn't give one to Mr. Wozniak just to use his name, 2 one meg Ramworks were paid for at full price).

9 IT'S FROM APPLIED ENGINEERING Unlike most of the competition, we only make accessories for Apple, so we'll never spend your money on IBM product research. Applied Engineering's years of experience and wide product line really pays off, and because of our high sales levels we buy most of our I.C. chips factory direct. So don't let our low prices fool you, they're caused by high volume production. That's why we can offer the most memory for the least money. Guaranteed!

10 IT'S GOT IT ALL

- ☑ Sharp 80 Column Text
- ☑ Double high resolution graphics (with or without RGB option)
- ☑ User Expandable to 1 Megabyte
- ☑ Can Use 64K or 256K RAMS in any combination
- ☑ Adds Memory to Appleworks
- ☑ Accelerates Appleworks
- ☑ 100% Compatibility with All IIe software
- ☑ RAM Disk software available, compatible with Applesoft, PRO-DOS, DOS 3.3, and PASCAL ($29)
- ☑ RAM Disk available for CP/M ($29). (This program is included with our CP/M card)
- ☑ Visicalc preboot available ($29)
- ☑ RGB option
- ☑ Takes only one slot
- ☑ 3 year no hassle warranty

11 THE PATENT OFFICE HAS ONE There are many advanced features on Ramworks, but two parts of the design are so advanced we applied for patents. One patent application deals with our ultra fast, ultra smooth 80 column screen display, and the other patent application deals with our ingenious way of dramatically reducing the power and heat of memory chips and improving reliability at the same time.

12 HERE TODAY, HERE TOMORROW In the seven years we've been making products for the Apple, we've seen a lot of companies come and go. Although nothing is forever, we're growing, expanding and we're profitable. And we are totally committed to Apple computing, which means you'll never run out of things to do with Ramworks. Or for that matter, reasons to buy one.

Ramworks™ with 64K	**$179**
Ramworks™ with 128K	**$249**
Ramworks™ with 256K . . .	**$299**
Ramworks™ with 512K . .	**$399**
Ramworks™ with 1 MEG	**$649**
RGB Option (can be added later)	**$129**

Call (214) 241-6060

9 a.m. to 11 p.m. 7 days a week or send check or money order to:
Applied Engineering, P. O. Box 798, Carrollton, Texas 75006

MasterCard VISA Visa and
C.O.D. welcome. No extra charge for credit cards. Texas Residents add 5½% sales tax. Add $10.00 if outside U.S.A.

APPLIED ENGINEERING
"We Set the Standard"

```
                        2830 *--------------------------------
096F- 00 00 03          2840 HEADER .DA 0,3,#$F0+VNSZ
0972- 00 F4
0974- 44 41 54          2850 VN     .AS /DATA/
0977- 41                2860 VNSZ   .EQ *-VN
04-                     2870        .BS 15-VNSZ
0978-
0983- 00 00 00
0986- 00 00 00
0989- 00 00 00
098C- 00 00 00          2880        .HS 00.00.00.00.00.00.00.00.00.00.00.00
098F- 00 00 C3
0992- 27 0D 00
0995- 00 06 00
0998- 08 00             2890        .HS 00.00.C3.27.0D.00.00.06.00.08.00
2B-                     2900 HDR.SZ .EQ *-HEADER
                        2910 *--------------------------------
                        2920 CLEAR.INIT.BUFFER
099A- A0 00             2930        LDY #0
099C- 98                2940        TYA
099D- 99 FC 09          2950 .1     STA INIT.BUFFER,Y
09A0- C8                2960        INY
09A1- D0 FA             2970        BNE .1
09A3- 60                2980        RTS
                        2990 *--------------------------------
                        3000 BOOTER
                        3010        .PH $800
                        3020 BOOTER.PHASE
0800- 01                3030        .HS 01
0801- BD 88 C0          3040        LDA $C088,X  MOTOR OFF
0804- A0 00             3050        LDY #0
0806- B9 14 08          3060 .1     LDA MESSAGE,Y
0809- F0 06             3070        BEQ .2
080B- 20 F0 FD          3080        JSR $FDF0
080E- C8                3090        INY
080F- D0 F5             3100        BNE .1
0811- 4C 59 FF          3110 .2     JMP $FF59
                        3120 *--------------------------------
                        3130 MESSAGE
0814- 8D 8D 87          3140        .HS 8D8D8787
0817- 87
0818- C3 CF CD
081B- C2 C9 CE
081E- C1 D4 C9
0821- CF CE A0
0824- C4 CF D3
0827- AF D0 D2
082A- CF C4 CF
082D- D3 A0 C4
0830- C1 D4 C1
0833- A0 C4 C9
0836- D3 CB             3150        .AS -"COMBINATION DOS/PRODOS DATA DISK"
0838- 8D 8D 87          3160        .HS 8D8D8787
083B- 87
083C- CE CF A0
083F- C4 CF D3
0842- A0 C9 CD
0845- C1 C7 C5
0848- A0 CF CE
084B- A0 D4 C8
084E- C9 D3 A0
0851- C4 C9 D3
0854- CB                3170        .AS -/NO DOS IMAGE ON THIS DISK/
0855- 8D 8D 00          3180        .HS 8D8D00
                        3190        .EP
                        3200 *--------------------------------
09FC-                   3210 INIT.BUFFER .BS 256
                        3220 *--------------------------------
0A02-                   3230 V.VOLUME    .EQ INIT.BUFFER-$BB+$C1
0A34-                   3240 V.BITMAP    .EQ INIT.BUFFER-$BB+$F3
                        3250 *--------------------------------
09FD-                   3260 C.TRACK        .EQ INIT.BUFFER+1
09FE-                   3270 C.SECTOR       .EQ INIT.BUFFER+2
                        3280 *--------------------------------
```

Software Sources for 65802 and 65816.....Bob Sander-Cederlof

Western Design Center reports rising interest among software developers in supporting the new 65802 and 65816 microprocessors.

Since the 65802 chip can be plugged into almost any old Apple, and 65816 co-processor cards are available for Apples, most new software is designed to run in Apples. Of course, the 65802 will also fit in old Ataris and Commodores and even the venerable KIM-1, but these are of lesser interest to AAL.

Four companies have adapted their Apple assemblers to include the new opcodes and addressing modes of the 65802 and 65816.

Of course, you know we have. Last December we released Version 2.0 of the S-C Macro Assembler, and in July we released the ProDOS version. Both of these include full support for the 6502, 65C02 (both standard and Rockwell versions), 65802, and 65816. The DOS version requires at least 48K RAM, and the ProDOS version requires at least 64K.

Other companies supporting the 658xx are Roger Wagner Publishing (Merlin Pro), The Byte Works (ORCA/M 3.5), and Lazerware (Lisa 3.2).

Merlin Pro is the latest version of Merlin, by Glen Bredon. (Big Mac, marketed by Call APPLE, is virtually the same as Merlin, not Merlin Pro.) Merlin Pro will only run in a //c or a //e with at least 128K RAM. In order to assemble the 65C02 additions, you must either be in a //c or in a //e with the enhanced monitor ROM. (If you have an older //e, you must first BRUN a file named MON.65C02.) 65816 support is not complete: the long 24-bit addressing modes were omitted on the premise that these are useless in a 65802 environment. (But what if I am developing code for a co-processor card with a 65816 on it?) The special opcodes in Rockwell's 65C02 are not directly supported, but a file of macro definitions is provided. Merlin Pro does include the capability of producing and linking relocatable object files with external symbols.

Lisa 3.2 is Randy Hyde's latest version of one of the fastest 6502 assemblers around. I have not seen 3.2, but it is reported to support the 65816.

ORCA/M (which is MACRO spelled backwards) was originally published by Hayden Software. They let it go after spending a lot of money promoting it as "the world's best assembler." I remember seeing that claim appear for the first time in Nibble magazine only a few pages away from the same claim in an ad for Nibble's own MicroSparc assembler. Anyway, ORCA/M is now published by The Byte Works, apparently connected more directly with the author (Mike Westerfield). ORCA/M was the first assembler to be revised to support the 65816, and as such Mike had the honor of deciding what some of the assembler rules and syntax would be.

David Eyes, author of the first book on 65816 assembly language, has developed a Pascal P-Code Interpreter which takes advantage of the 65802 features and works with Apple Pascal. (191 Parkview Ave., Lowell, MA 01852)

Starlight Forth Systems has a FIG Forth compatible package for the 802/816, for operation in an Apple. (15247 North 35 St., Phoenix, AZ 85032)

Comlog offers an Applesoft compatible, extended Basic which can be used in an Apple //e equipped with their 65816 co-processor board. (7825 E. Redfield Rd, Scottsdale, AZ 85260)

Manx Software claims to have a 65816 C compiler and assembler under development. (Box 55, Shrewsbury, NJ 07701)

Will Troxell, of MicroMagic, is not only developing a co-processor card for Apples. He is also producing the first operating system for the 65816, which will be similar to Unix.

Problems with 65802's in Apple II+.....Bob Sander-Cederlof

Much to our dismay, we have just learned that some Apple II+ machines will not function properly with a 65802 installed. It is probably the same kind of timing problem that exists with the 65C02 in nearly all Apple II and Apple II+ machines. We had thought the 65802 would work in all II and II+ machines, but it will not. It works in my old II, and one of my II+ machines, but not the other one. We have heard of lots of successful installations, and a few unsuccessful ones. We have not yet heard whether changing to 74F257's will fix things up, as it does with the 65C02.

If you would like to try this exciting enhancement in your Apple, we are selling the 2 MHz 65802 for only $50 (plus $1.50 shipping, and plus 6.125% sales tax if you are in Texas). (The price direct from Western Design Center is still $95 each.) If you want to try it in a II or II+, go ahead and order one; if it turns out to be incompatible, you can send it back for a refund.

I hope we are safe in assuming that anyone who orders such a chip knows how to properly handle, install, and remove CMOS parts. They are extremely sensitive to static electricity, at levels too small for humans to even feel. You can kill them with the voltage generated by moving your arm, if you are wearing a synthetic shirt. You need to be careful, very careful. It is also very easy to bend the leads, or insert the parts backwards. I know, because I have done it. If you want a 65802 but are not confident about the installation, find someone who will do it for you.

Short Binary-to-Decimal Conversion in 65802.....Bob S-C

Since the 65802 supports 16-bit registers, it is possible to write a very tiny loop that will convert 16-bit binary numbers to four- or five-digit decimal values. Jim Poponoe called today and suggested the idea to me.

The idea is to count down the binary number in binary mode while incrementing a four-digit decimal value in the A-register It certainly isn't very fast, but it is small.

The two programs below illustrate the technique. CONV.1 converts a two-byte value at $0000 (and $0001) and stores the four-digit result in $0002 (and $0003). CONV.1 goes one step further, handling a fifth digit which is stored in $0004.

You could use CONV.1 inside the CATALOG command to convert volume numbers and sector counts. It is probably shorter than the existing code. Since the numbers converted are less than 500, the maximum time is still less than half a millisecond.

Lines 1080 and 1090 put the 65802 into "native" mode, so that we can use the 16-bit features. Lines 1210,1220 put the 65802 back into 6502 "emulation" mode, since the subroutine was written under the assumption that the caller would be in emulation mode. If you plan to use the subroutine within a program that runs entirely in native mode, you could leave these four lines out. If you plan to call it from both native mode and emulation mode, you need to save the E status and restore it at the end. You can do that like this:

```
CONV.1 CLC      ENTER NATIVE MODE
       XCE
       PHP      SAVE CALLER'S MODE (IN C-BIT)
        .
        .
        .
       PLP      GET CALLER'S MODE
       XCE      RESTORE CALLER'S MODE
       RTS
```

Line 1100 clears both the X- and M-bits, so that all 16-bit features are on. Note that when either of these bits are cleared, immediate-mode operands are two bytes long. The assembler doesn't keep track of the state of these two bits, because it would be impossible in the general case without a complete flow analysis of the program. It is up to the programmer to tell the assembler whether to assemble one- or two-byte immediate operands. You do this in S-C Macro Assembler by using a double pound-sign notation, as in lines 1110 and 1160.

Line 1110 loads a full 16-bit value zero into the A-register. Line 1120 loads the 16-bit value from location $0000 and $0001. the low byte of the value is in $0000, and the high byte in $0001. If all 16-bits of this value are zero, line 1130 will branch around the conversion loop. If not, it will not branch.

DISASM 2.2e - AN INTELLIGENT DISASSEMBLER : $30.00

Investigate the inner workings of machine language programs. DISASM converts machine code into meaningful, symbolic source. Creates a standard text file compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even lets you substitute your own meaningful labels. (100 commonly used Monitor and Pg Zero names included.) An address-based triple cross reference table is provided to screen or printer. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his new ASSEMBLY COOKBOOK. For entire Apple II family including the new Apple //c (with all the new opcodes). SOURCE CODE available for an additional $30.00

LOW LOW PRICE !!! C-PRINT For The APPLE //c : $69.00

Connect standard parallel printers to an Apple //c. C-PRINT is a hardware accessory that plugs into the standard Apple //c printer serial port. The other end plugs into any printer having a standard 36 pin centronics-type parallel connector. Just plug in and print! High speed data transfer at 9600 Baud. No need to reconfigure serial port or load software drivers for text printing.

FONT DOWNLOADER & EDITOR : $39.00

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. All special printer functions (like expanded, compressed etc.) apply to custom fonts. Full HIRES screen editor lets you create your own characters and special graphics symbols. Compatible with many parallel printer I/F cards. User driver option provided. For Apple II, II+, //e. Specify printer: Apple Dot Matrix, C.Itoh 8510A (Prowriter), Epson FX 80/100, or OkiData 92/93.

The Font Downloader & Editor for the Apple Imagewriter Printer. For use with Apple II, II+, //e (with SuperSerial card) and the new Apple //c (with builtin serial interface).

FONT LIBRARY DISKETTE #1 : $19.00 Contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

The 'PERFORMER' CARD : $39.00

Plugs into any slot to convert a 'dumb' centronics-type printer I/F card into a 'smart' one. Command menu eliminates need to remember complicated ESC codes. Features include perforation skip, auto page numbering with date & title. Includes large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Itoh 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93. SOURCE CODE: $30.00

FIRMWARE FOR APPLE-CAT: The 'MIRROR' ROM : $25.00

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support. Uses superset of Apple's Comm card and Micromodem II commands. SOURCE CODE: $50.00

RAM/ROM DEVELOPMENT BOARD : $30.00

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into $Cn00-CnFF and $C800-CFFF.

ALL NEW !!! MIDI *MUSIC PRODUCTS*

MIDI means Musical Instrument Digital Interface. Use your computer with any MIDI-equipped music keyboard for entertainment and music education. Low cost MIDI player interface cable, complete with 6 song demo disk: $49.00. Thousands of popular songs available soon on diskette (also compatible with Passport MIDI interface). Products for both the Apple IIc and Commodore 64/128. Unique general purpose MIDI expander cable and gender changer also available. Send SASE for product descriptions and prices.

Avoid a $3.00 handling charge by enclosing full payment with order. VISA/MC and COD phone orders accepted.

RAK-WARE 41 Ralph Road W. Orange N J 07052 (201) 325-1885

Line 1140 sets the decimal mode, which affects only the ADC and SBC instructions. Line 1190 turns it back to binary. If you use the PHP and PLP steps shown above in the discussion about native and emulation modes, you could leave out the CLD in line 1190: the PHP would restore the D-bit properly.

The loop in lines 1160-1180 adds one to the A-register and subtracts one from the X-register, until the X-register reaches zero. Since we are in decimal mode, the A-register counts up in BCD format. The largest number the loop can handle correctly is 9999 decimal ($270F). Larger values will not even have the correct lower four digits, since CARRY gets set when 9999 is incremented.

After the loop finishes, line 1200 stores the result low-byte-first at $0002 and $0003.

CONV.2 is almost identical to CONV.1, on purpose. There are five new lines of code, at lines 1330, 1390-1410, and 1480. We use the Y-register to keep track of the fifth digit, so that we can convert numbers larger than 9999. Line 1330 sets Y=0. Line 1390 checks for the carry that occurs when 9999 is incremented. If there is no carry, the loop is the same as in CONV.1. If there is a carry, line 1400 increments the Y-register and line 1410 clears carry. (We could save one byte at the expense of slower operation by including the CLC on line 1370 inside the conversion loop.)

Line 1480 stores the fifth digit in location $0004. I put it after the switch back to emulation mode, since I only wanted to store one byte.

I timed these subroutines by counting cycles, as shown in the comments in lines 1040,1050 and 1250,1260. In the process I was suprised to learn that the DEX opcode still takes only two cycles, even when in 16-bit mode. Of course, the same goes for INX, DEY, INY. It is also true of ASL, LSR, ROL, ROR, INC, and DEC when the operand is the A-register.

```
                1000 *SAVE S.65802.CONVERSIONS
                1010 *--------------------------------
                     1020         .OP 65802
                     1030 *--------------------------------
                     1040 *   CONVERT UP TO 9999, MAX TIME < 80 MSEC
                     1050 *      # CYCLES = 8*NUMBER + 44
                     1060 *--------------------------------
                     1070 CONV.1
000800- 18           1080         CLC            ENTER 65802 NATIVE MODE
000801- FB           1090         XCE
000802- C2 30        1100         REP #$30       16-BIT MODES
000804- A9 00 00     1110         LDA ##0        START WITH 0000
000807- A6 00        1120         LDX 0          GET NUMBER TO BE CONVERTED
000809- F0 09        1130         BEQ .2         ...IT IS 0000
00080B- F8           1140         SED            ENTER DECIMAL MODE
00080C- 18           1150         CLC
00080D- 69 01 00     1160 .1      ADC ##1        INCREMENT BCD VALUE
000810- CA           1170         DEX            DECREMENT BINARY VALUE
000811- D0 FA        1180         BNE .1         ...NOT FINISHED YET
000813- D8           1190         CLD            BACK TO BINARY MODE
000814- 85 02        1200 .2      STA 2          STORE RESULT
000816- 38           1210         SEC            BACK TO 6502 EMULATION MODE
000817- FB           1220         XCE
000818- 60           1230         RTS            RETURN TO CALLER
```

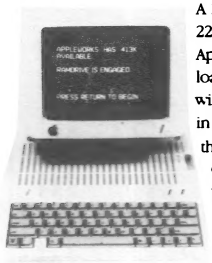
Expanding Your IIc Is Easy With Z-RAM

Applied Engineering and Apple computer have teamed up to take your IIc to new heights.

Applied Engineering's Z-RAM card for the IIc is available with 256K or 512K of additional memory and a powerful Z-80 microprocessor for running CP/M software.

Z-RAM fits neatly inside the IIc. Installation is easy, clear instructions show you how. You'll need a screwdriver and about 10 minutes (if you can change a light bulb you can install Z-RAM).

Z-RAM and Appleworks will knock your socks off.



A 256K Z-RAM will give you a 229K available desktop and Appleworks will be completely loaded into memory. Appleworks will now run about 10 times faster in your IIc with 1 disk drive than in other IIc's with 2 disk drives. A 512K Z-RAM will give you a 413K available desktop. A 256K Z-RAM can be upgraded to 512K by just plugging in more memory chips.

Z-RAM is also a high speed solid state disk drive. With Z-RAM, your programs will load and save over 20 times faster. Z-RAM's RAM disk is compatible with Applesoft, Pro-DOS, DOS 3.3, PASCAL and CP/M. And with Z-RAM, you can copy a disk in one pass. Just insert the original, remove the original, insert blank disk! That's it! Z-RAM is another disk drive, only 20 times faster, 4 times larger capacity, and no whirring, clicking or waiting!

But before you start panting over all that extra memory, don't forget that the Z-RAM card has a built-in high speed Z-80 processor chip that allows you to run CP/M programs like Wordstar, dBASE II, Turbo PASCAL, Microsoft BASIC, FORTRAN and COBOL and over 3,000 other CP/M programs. So Z-RAM not only makes Apple programs run better and faster, it lets you run MORE programs.

With the Z-RAM card installed, your IIc is still your IIc only now you'll have that extra memory that Appleworks and other programs need. And you can run all that great CP/M software that others can only dream about.

Z-RAM is 100% compatible with all IIc software and hardware including the mouse, 2nd disk, modem and printer. Z-RAM is easily handled by the IIc power supply as power consumption is kept very low by using two custom integrated circuits and a patent pending power saving design. And Z-RAM is from Applied Engineering, the acknowledged leader and innovator of accessories for the Apple.

Z-RAM comes complete with manual, RAM disk software, Z-80 operating system, CP/M manual and a 3 year no hassle warranty.

So the next time somebody asks you why you didn't get an IBM P.C., tell him you bought a IIc because the IBM didn't have enough memory and was too slow and couldn't run CP/M software. And tell him you made it past the 8th grade.

Z-RAM with 256K **$449**
Z-RAM with 512K **$549**

If you want to run CP/M software, but don't need more memory, may we suggest our Z-80c card. The Z-80c offers the same CP/M performance as Z-RAM but has no memory expansion ports. And the Z-80c will not affect the running of Apple programs. The Z-80c is priced at only $159.00 and should you ever want to upgrade to Z-RAM, we'll refund your full purchase price.

Call (214) 241-6060
9 a.m. to 11 p.m. 7 days a week or

Send check or money order to:
Applied Engineering
P. O. Box 798
Carrollton, Texas 75006

MasterCard Visa and C.O.D. welcome. No extra charge for credit cards. Texas residents add 5⅛% sales tax. Add $10.00 if outside U.S.A.

APPLIED ENGINEERING
"We Set the Standard"

```
                        1240 *--------------------------------
                        1250 *   CONVERT UP TO 65535, MAX TIME < 705 MSEC
                        1260 *     # CYCLES = 11*NUMBER +3*INT(NUMBER/10000) + 50
                        1270 *--------------------------------
                        1280 CONV.2
000819- 18              1290        CLC             ENTER 65802 NATIVE MODE
00081A- FB              1300        XCE
00081B- C2 30           1310        REP #$30        16-BIT MODES
00081D- A9 00 00        1320        LDA ##0         START WITH 0000
000820- A8              1330        TAY             CLEAR 10000'S DIGIT
000821- A6 00           1340        LDX 0           GET NUMBER TO BE CONVERTED
000823- F0 0D           1350        BEQ .2          ...IT IS 0000
000825- F8              1360        SED             ENTER DECIMAL MODE
000826- 18              1370        CLC
000827- 69 01 00        1380 .1     ADC ##1         INCREMENT BCD VALUE
00082A- 90 02           1390        BCC .3
00082C- C8              1400        INY             INCREMENT 10000'S DIGIT
00082D- 18              1410        CLC
00082E- CA              1420 .3     DEX             DECREMENT BINARY VALUE
00082F- D0 F6           1430        BNE .1          ...NOT FINISHED YET
000831- D8              1440        CLD             BACK TO BINARY MODE
000832- 85 02           1450 .2     STA 2           STORE RESULT
000834- 38              1460        SEC             BACK TO 6502 EMULATION MODE
000835- FB              1470        XCE
000836- 84 04           1480        STY 4           STORE 10000'S DIGIT
000838- 60              1490        RTS             RETURN TO CALLER
                        1500 *--------------------------------
```


Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is $18 per year in the USA, sent Bulk Mail; add $3 for First Class postage in USA, Canada, and Mexico; add $14 postage for other countries. Back issues are available for $1.80 each (other countries add $1 per back issue for postage).